서버 엔진, 온라인게임 구축을 위한 핵심 기술

✻ 클라이언트-서버 구조의 온라인게임의 안정적인 네트워킹 가능케 해

한국전자통신연구원(ETRI)
김현빈 외 가상현실연구부

연 재 순 서

1. 3D 게임엔진에 대하여
2. 렌더링 엔진
3. 애니메이션 엔진
4. 사운드 엔진
5. 서버 엔진(이번호)
6. 게임 인공지능
7. 맵 에디터
8. 게임진행 모듈

'서버 엔진'이란 네트워크를 통한 대규모 사용자들 간의 상호작용이 요구되는 게임(대표적인 것으로 클라이언트-서버 구조를 갖는 온라인게임)을 구현하기 위해 필요한 핵심적인 기능인 효율적이고 안정적인 네트워킹 기능 외에 기타 다양한 성능 최적화 기능들과 관련 API들을 제공하는 엔진이다.

일반적으로 온라인게임에서 게임 서버는 서로 다른 특성을 갖는 수천, 수만 명의 클라이언트들과 메시지를 주고받으며, 받은 메시지들을 실시간으로 처리해서 그 결과를 관련 클라이언트들에게 되돌려 주어서 모든 클라이언트들이 일관된 데이터들을 가지도록 해주어야 한다. 그리고 게임 서버들은 24시간 365일 중단 없이 서비스를 계속해야 하므로 무엇보다도 안정적으로 실행되어야 할 뿐만 아니라 서버의 과부하, 허용치 이상의 네트워크 지연 등으로 인한 게임 성능 저하를 막아 서버의 성능을 최적으로 유지시킴으로써 사용자들의 불편을 최소화해야 한다.

이와 같은 온라인게임 서버들을 구현하기 위해서는 많은 어려움을 겪게 되고 많은 시간이 요구된다. 또한, 구현을 했더라도 이를 테스트하기 위해서는 수천, 수만 명의 클라이언트들이 접속하는 환경을 만들어야 하기 때문에 구현 후 테스트를 위해서도 추가로 많은 비용과 시간이 들게 된다.

이런 어려움을 극복하기 위해서 필요한 것이 서버 엔진이다. 서버 엔진에서는 가장 핵심적인 기능으로서 클라이언트와 서버간의 효율적이고 안정적인 네트워킹을 위해서 중간 단계에서 필요한 각종 작업들을 처리해주는 기능을 제공해 주어야 한다. 이외에도 서버 성능의 최적화를 위해 여러 가지 다양한 기능들을 제공해주어야 한다. 그러면 효율적인 서버 엔진이 되기 위해 사용될 수 있는 최적화 기법들에 대해 알아보기로 하자.

1. 관심영역(AOI: Area Of Interest) 관리

서버는 공유상태의 변화를 사용자에게 알릴 필요가 있을 때 그 변화가 일어난 곳 주위의 특정 영역 내에 있는 사용자에게만 공유상태를 보냄으로써 공유상태의 전체 전송률을 크게 줄일 수 있다. 이때 공유상태를 보내는 영역을 관심영역이라 한다.

그림 1
거리 기반의 관심영역

관심영역 관리에는 두 가지 방법이 있다. 〈그림 1〉과 같이 거리를 기반으로 할 경우에는 특정 거리 내에 있는 사용자에게만 공유상태를 보낸다. 이 경우 공유상태의 전송율은 줄일 수 있지만, 공유상태의 변화가 있을 때마다 전체 사용자들과의 거리를 검사해야 하므로 서버의 부하가 많이 늘어날 수도 있다. 그러므로 전체 사용자들과의 거리를 검사하지 않도록 하는 방법이 필요하다.

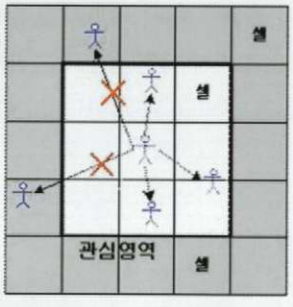

그림 2
셀 기반의 관심영역

〈그림 2〉는 셀 기반의 관심영역 관리 방법이다. 게임서버에서 게임 공간을 셀이라고 하는 단위영역들로 나누어서 어느 특정 셀에서 발생된 공유상태의 변경을 그 주위에 있는 셀들 내의 사용자들에게만 보내주는 것이다. 그러면 모든 사용자들의 거리를 일일이 검사할 필요없이 공유상태를 보내야 하는 사용자에게 바로 보낼 수 있다. 이를 위해서 게임서버는 사용자, 몬스터 등과 같은 공유객체들을 셀 별로 따로 관리하고 있어야 하며, 각각의 셀들은 자신 주위의 셀들에 대한 리스트를 가지고 있어야 한다.

위의 거리 기반의 관심영역 관리와 셀 기반의 관심영역 관리를 혼용해 공유상태의 변경이 일어난 셀 주위의 셀들 내에 있는 사용자들만 거리를 검사하여 특정 거리 내에 있으면 공유상태를 보내줄 수도 있다.

2. 공유 상태 관리 기법들

게임 상에서 수많은 사용자들은 다양한 동적인 상태들을 서로 공유하고 있기 때문에 이런 공유상태들을 일관성있게 유지시켜 줄 필요가 있다. 하지만, 네트워크 상에서 다양하게 분포되어 있는 여러 클라이언트 시스템과 서버 시스템 사이에 네트워크 지연과 대역폭의 제한으로 인해 모든 사용자들의 공유상태들이 모두 동일하도록 완벽하게 일관성을 유지시켜주는 것은 거의 불가능하다.

일반적으로 눈으로 보이는 화면상에 대하여 거의 완벽하게 일관성을 유지하기 위해서는 초당 30번 이상 공유상태들을 주고받아야 한다. 이럴 경유 서버는 네트워크 대역폭의 제한 때문에 많은 사용자들을 수용해줄 수 없을 것이다. 반면에, 공유상태들의 전송율을 지나치게 줄이면 게임서버는 많은 사용자들을 수용할 수는 있겠지만 일관성이 실시간으로 유지되지 않으므로 사용자들이 서로 다른 상태를 가지게 되어 게임을 제대로 진행해 나갈 수 없게 된다.

이와 같이 서버 성능(서버의 최대 수용가능 동시사용자 수)과 게임상태의 일관성은 둘 다 높을수록 좋지만 서로 역비례 관계에 있기 때문에 양쪽을 모두 고려해 공유상태들의 전송율을 적절히 유지시켜줄 필요가 있다. 그러한 관점에서 동적인 공유상태의 관리방법에는 다음과 같은 세 가지 방식이 쓰인다.

(1) 중앙 관리 기법

이 방법은 중앙에 공유상태 저장소가 있어서, 토큰 패싱 기법을 사용해 한번에 한 사용자만이 공유상태 저장소에 접근해 상태를 변경시킬 수 있도록 한다. 그리고 상태가 변경되면 그것을 모든 사용자들에게 전달해 확인을 받는다. 이 방법은 일관성을 확실하게 유지시켜줄 수 있는 반면에 성능(최대사용자 수, 지연 등)에 문제가 있기 때문에 실시간성이 요구되거나 공유상태의 변화가 빈번하거나 동시 사용자가 많은 경우에는 사용이 어렵다.

(2) 빠른 상태 갱신

이것은 공유상태에 대한 소유권을 가지고 있는 사용자가 상태변화의 유무에 상관없이 매우 짧은 주기로 다른 모든 사용자들에게 계속해서 상태갱신 정보를 보내는 방법이다. 받는 쪽에서는 받은 것에 대한 확인은 하지 않는다. 이 방법은 일관성을 확실하게 보장해줄 필요가 없으며 비교적 소수의 사용자들을 위한 곳에 쓰인다.

(3) 데드 레커닝 알고리즘

데드레커닝(dead reckoning) 알고리즘에서 공유상태를 변화시킨 사용자는 다른 사용자들에게 특정조건이 만족될 때에만 변화된 상태정보를 보내준다. 그리고 변화된 상태정보를 받는 사용자들은 불연속적으로 받은 상태정보를 이용해 연속적인 값을 만들어 사용한다.

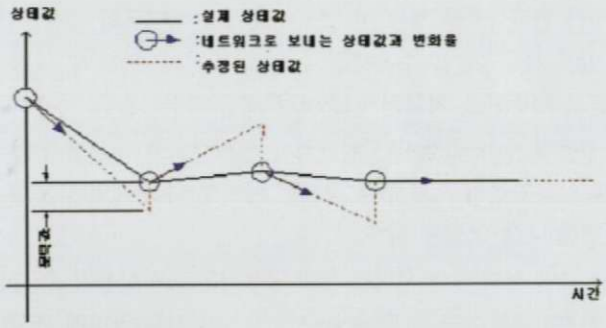

그림 3 상태정보를 보내는 사용자에서의 알고리즘

공유상태를 변화시키는 사용자(공유상태를 보내는 사용자)는 과거에 보낸 상태정보(상태값, 변화율 등)를 근거로 해 추정 알고리즘(tracking algorithm)을 이용해 다른 사용자들이 추정하고 있을 공유상태를 계속해서 추정한다. 그리고 계속해서 추정된 상태값과 실제 상태값과의 오차를 검사해 이 오차가 미리 정해진 문턱값 이상이 되면 다른 사용자들에게 새로운 상태정보(상태값, 변화율 등)를 보낸다. 따라서 불필요한 상태정보의 전송을 막을 수 있으며 문턱값을 조정해줌으로써 상태정보의 전송율을 제어해줄 수도 있다. 문턱값이 크면 클수록 사용자들 간의 공유상태 오차는 커지지만 상태정보의 전송율은 떨어진다. 반면에 문턱값이 작으면 작을수록 공유상태의 오차는 줄지만 전송율이 늘어나게 된다. 한편, 다른 사용자들과의 거리에 따라 사용자별로 문턱값을 달리해주기도 한다. 즉, 가까이에 있는 사용자에 대해서는 문턱값을 작게 해 공유상태를 보다 자주 보내서 공유상태의 오차를 줄이고 멀리 있는 사용자에 대해서는 문턱값을 크게 해 상태정보의 전송율을 줄일 수도 있다.

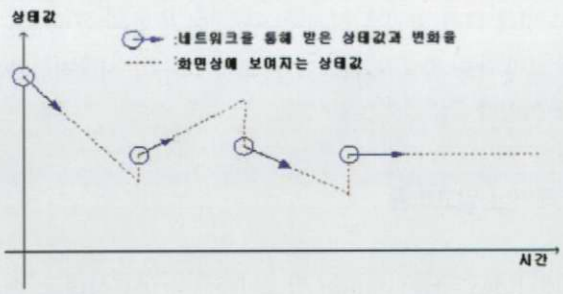

그림 4 공유상태를 받는 사용자에서의 알고리즘

한편, 변화된 공유상태를 받는 사용자는 과거에 받은 공유상태를 근거로 해 현재의 공유상태를 계속해서 추정하다가 새로운 상태정보를 받으면 공유상태를 보정해준다. 이때 〈그림 4〉와 같이 불연속적인 상태값이 변하는 것을 막기 위해 여러 가지 수렴 알고리즘(convergence algorithm)들을 사용해 공유상태를 갱신해준다.

3. 다중/분산 서버

온라인게임의 사용자수는 계속해서 증가되고 있는 추세이다. 수천명에서 수십만 명으로 멀지 않은 미래에는 전체 인구 모두가 교육, 오락, 의료 등의 목적으로 여러 가지 형태의 온라인 콘텐츠에 접속하게 될지도 모른다. 그러나 현재 하나의 게임서버가 수용할 수 있는 동시 사용자는 고작해야 수 천명 정도의 수준이다. 그러므로, 계속해서 증가하는 대규모의 온라인 게임 사용자들을 지원해주기 위해서는 게임서버를 지속적으로 늘려가야 한다.

(1) 다중서버 방식

이 방식은 〈그림 5〉와 같이 동일한 게임공간을 독립적으로 여러 개를 만들고 그 각각을 서비스하는 게임서버를 두는 것이다.

그림 5 다중서버 방식

이 방식은 게임서버의 수만큼 사용자수를 증가시킬 수 있는 반면에 서로 다른 서버에 있는 사용자들 간의 상호작용이 불가능하다. 그렇기 때문에 실질적으로 단일 게임 공간상에 있는 동시 사용자수는 늘어나지는 않고, 구조가 단순해 손쉽게 사용

자수를 늘릴 수 있다. 현재 서비스를 하는 있는 많은 온라인게임들이 이와 같은 형태로 서비스를 하고 있다.

(2) 분산서버 방식

이 방식은 하나의 게임공간을 여러 개의 서버가 분담해 서비스하는 방식이다. 서비스의 분담은 게임공간 내의 사용자를 기준으로 여러 개의 그룹을 만들어서 각각의 사용자 그룹에 서버를 두는 사용자 기반의 분산서버 방식이 있고, 게임공간 자체를 여러 개의 영역으로 분할하여 각각의 영역을 담당하는 게임서버를 따로 두는 영역기반의 분산서버 방식이 있다.

사용자 기반의 분산서버 방식에서는 부하를 분산하기가 쉬운(간단한 예로 새로운 사용자를 최대 사용자수에 도달하지 않은 서버로 연결시켜 줌) 반면, 서버들 간의 통신량이 급격히 증가하므로 서버들을 모두 근거리에 모아 두고 서버들간을 외부와 독립된 내부 로컬 랜으로 연결해야 한다. 그렇게 하더라도 계속해서 확장해 나가는 제약이 따르게 된다. 그리고 다른 서버가 관리하는 사용자에 대한 상태정보는 서버를 한번 더 거쳐야 하기 때문에 지연문제가 발생할 수 있어서 일반적으로 온라인게임에서는 영역기반의 분산서버 방식을 많이 쓰고 있다.

영역 기반의 분산서버 방식은 다시 두 가지 방식으로 나눌 수 있는데, 첫 번째 방식은 〈그림 6〉과 같이 게임공간내의 여러 영역들이 서로 단절돼 있어서 영역간의 상호작용은 불가능하지만 특정 포탈(portal)을 통해서 다른 영역으로 이동이 가능한 방식이다.

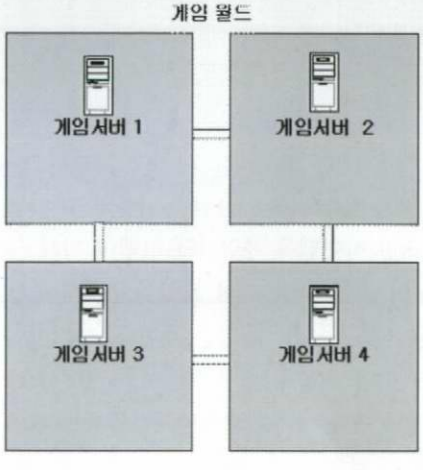

그림 6 포탈로 연결된 영역기반 분산서버

이 방식은 다중서버 방식에서 쉽게 확장이 가능하기는 하지만 영역간의 상호작용이 불가능하기 때문에 다중서버 방식에 비해 큰 잇점이 없어, 현재 많이 사용되지는 않고 과도기적으로 몇몇 곳에서 사용되고 있다.

두 번째 방식은 지금까지 가장 진보된 방식으로서 〈그림 7〉과 같이 게임공간내의 분할된 영역들이 서로 끊김없는(seamless) 하나의 게임공간을 구성하는 방식이다.

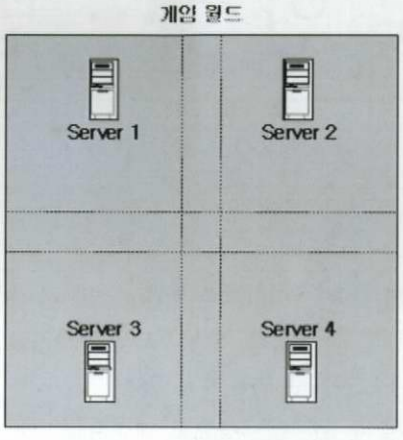

그림 7 끊김없는(seamless) 영역기반의 분산서버 방식

이 방식은 이전의 방식에 비해 영역들 간의 상호작용이 아무런 제한없이 가능하며 사용자들은 서로 다른 서버가 담당하는 영역들 간을 자유롭게 이동 가능하다. 사용자들은 자신이 다른 영역으로 넘어갈 때 그것을 전혀 인지하지 못할 수도 있다(경우에 따라 약간의 지연이 생길 수도 있다). 그리고 이 방식은 서버를 계속 늘림으로써 게임공간을 확장할 수 있을 뿐 아니라 게임공간 내의 허용 가능한 동시 사용자수도 필요한 만큼 늘릴 수 있다. 반면에 이 방식은 영역들이 서로 인접하고 있는 경계부분에서의 처리가 매우 복잡하고 하나의 영역으로 사용자들이 지나치게 집중되는 현상도 있을 수 있으므로 부하 분산을 충분히 고려해주어야 한다. 이와 같은 어려움에도 불구하고 많은 주요 온라인게임들이 향후에 이 방식을 사용하려 하고 있고, 몇몇은 이미 사용 중에 있다.